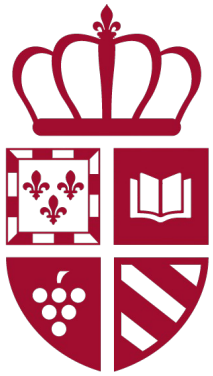

UNIVERSITÉ BOURGOGNE EUROPE

Polytech Dijon

FISA IOT 5A

Compte rendu projet VHDL

Controlleur SRAM

Auteur :
HELLE Evan

Enseignant :
DUBOIS Julien

POLYTECH DIJON

2025-2026

Sommaire

Table des Figures

Ce document a été réalisé avec LaTeX.

1 Compte Rendu

1.1 Introduction

Ce projet consiste à concevoir et implémenter en VHDL un contrôleur pour une mémoire SRAM de type mt55l512y36f, dont le modèle VHDL est fourni par le fabricant.
L'objectif est de permettre les opérations de lecture et d'écriture tout en respectant les contraintes temporelles et les spécifications électriques du composant.

Le contrôleur recevra les données via une interface d'entrée-sortie (IO Buffer) et assurera la communication avec la mémoire SRAM à travers une interface dédiée.
Son fonctionnement sera structuré autour d'une machine à états finis (FSM) afin de gérer de manière séquentielle et fiable les différentes phases d'accès mémoire.

Le projet inclura également la réalisation de `test bench` destinés à valider le comportement et la conformité de chaque élément du contrôleur.

1.2 IO Buffer

1.2.1 Fonctionnement

Un IO Buffer est un composant matériel utilisé dans les circuits pour gérer les signaux d'entrée et de sortie entre le circuit et le monde extérieur.
L'implémentation de L'IO Buffer que nous allons utilisé nous nous a été fourni.

Voici la représentation schématique de l'IO Buffer que nous allons utilisé dans notre projet :

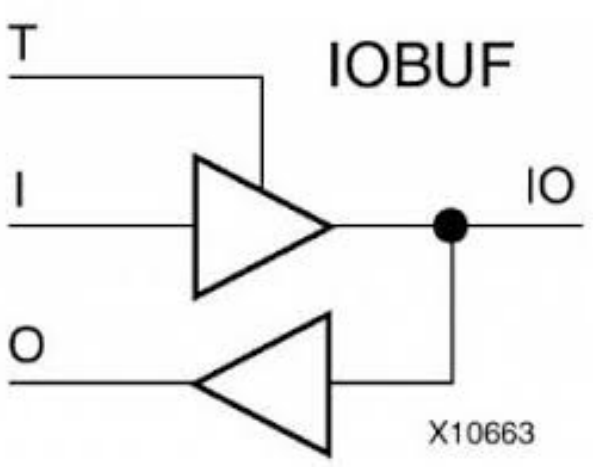

Figure 1.1: IO buffer

L'IO Buffer est composé de plusieurs entrées et sorties :

- **I** : Donnée d'entrée (input data)
- **O** : Donnée de sortie (output data)
- **T** : Trigger, permet de contrôler le mode de l'IO Buffer (entrée ou sortie qui servira à gerer l'ecriture ou la lecture dans notre cas)
- **IO** : Bus de données bidirectionnel (data bus)

1.2.2 Test Bench

L'objectif de faire un test bench est de vérifier le bon fonctionnement de l'IO Buffer avec la SRAM.
De plus cela premettra de de comprendre son fonctionnement avant de l'intégrer dans le projet final.
Il est important de faire ce test car il est nécessaire d'avoir un décalage entre la donnée et les signaux de contrôle au niveau de la SRAM.

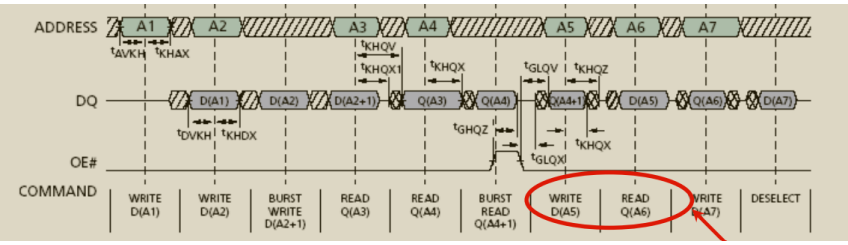

Figure 1.2: Extrait ennoncé projet

D'après l'ennoncé du projet et la datasheet de la SRAM, il est nécessaire d'avoir un décalage d'un cycle d'horloge entre la donnée et les signaux de contrôle.

Code

Pour le code du test bench, nous sommes partie sur une ecriture de donnée à l'adresse 1 et 2, puis une lecture de ces deux adresses.

```
tb : PROCESS
BEGIN
-- init
nCKE    <= '0';
nADVLD  <= '0';
nOE     <= '0';-- output enable
nCE     <= '0';
nCE2    <= '0';
CE2     <= '1';
SA      <= (others => '0');
Trig    <='1'; -- se mettre en "lecture" le temps de l'init pour ne pas ecrire n'importe quoi

wait for 6*(TCLKL+TCLKH);
```

```
SA                  <= "000"&x"0001";
Trig    <= '0'; -- ecriture à l'adresse 1
wait for 1*(TCLKL); -- pour travailler sur front montant
ENTREE  <= (others => '1'); -- decalage de la donnée d'un cycle
                            --par rapport à l'adresse et la commande


wait for 2*(TCLKL+TCLKH);
SA                  <= "000"&x"0002"; -- eriture à l'adress 2
wait for 1*(TCLKL+TCLKH);
ENTREE  <= ENTREE + 1;-- decalage de la donnée d'un cycle par
                      -- rapport à l'adresse et la commande

wait for 2*(TCLKH+TCLKL);
SA                  <= "000"&x"0001"; -- lecture à l'adresse 1
Trig <= '1';
wait for 2*(TCLKH+TCLKL);
SA                  <= "000"&x"0002"; -- lecture à l'adresse 2

wait; -- will wait forever
END PROCESS;
```

Ici la donnée est bien décalée d'un cycle par rapport à l'adresse et la commande. Quand le trigger est à 0, on est en écriture, et quand il est à 1, on est en lecture. C'est le cas pour la SRAM et l'IO buffer. Donc nous avons connecter par un fil ces deux I/O.

```
SRAM1 : mt551512y36f port map
(DQ, SA, '0', CLKO_SRAM, nCKE, nADVLD, '0',
'0', '0', '0', Trig, nOE, nCE, nCE2, CE2, '0');

IOb: for I in 0 to 35 generate
Iobx: IOBUF_F_16  port map(
O => SORTIE(I),
IO => DQ(I),
I => ENTREE(I),
T => Trig
);
end generate;
```

Simulation

Voici le résultat de la simulation :

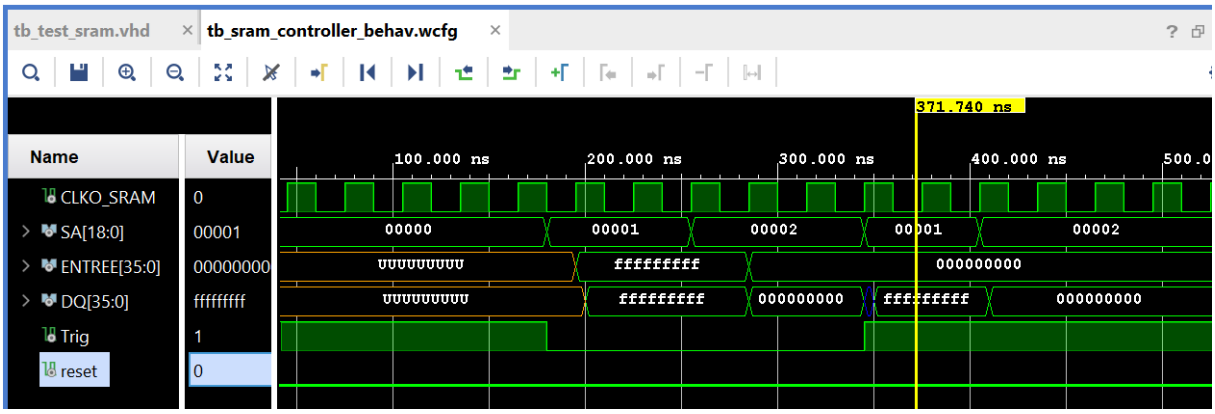

Figure 1.3: Simulation IO Buffer

Les données sont bien écrites aux bonnes adresses (1 et 2) et lues correctement par la suite.
Au passage à la lecture, le bus bidirectionnel passe en haute impédance, ce qui est le comportement attendu.
Voici un zoom sur le passage en haute impédance :

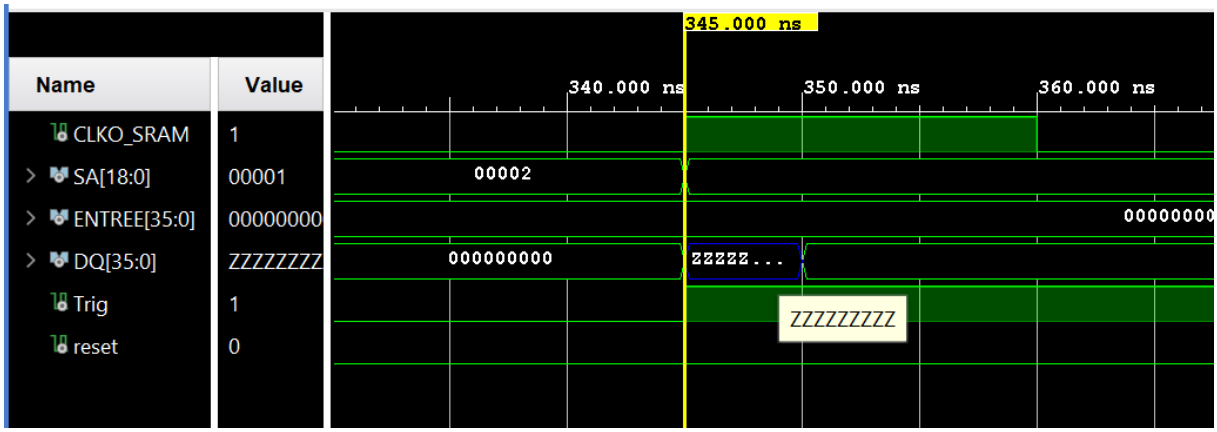

Figure 1.4: Haute impédance à la lecture de la donnée

1.3 Crontrolleur SRAM

Maintenant que nous avons validé le bon fonctionnement de l'IO buffer connecté à la SRAM, nous allons implémenter le contrôleur de la SRAM. Celui-ci sera responsable de la gestion des opérations de lecture et d'écriture vers la mémoire SRAM en fonction des signaux d'entrée.
Son fonctionnement sera basé sur une machine à états finis (FSM) qui gérera les différents états nécessaires pour effectuer les opérations de lecture et d'écriture.
un IO buffer sera également présent pour interfacer les données entre le contrôleur et la SRAM.

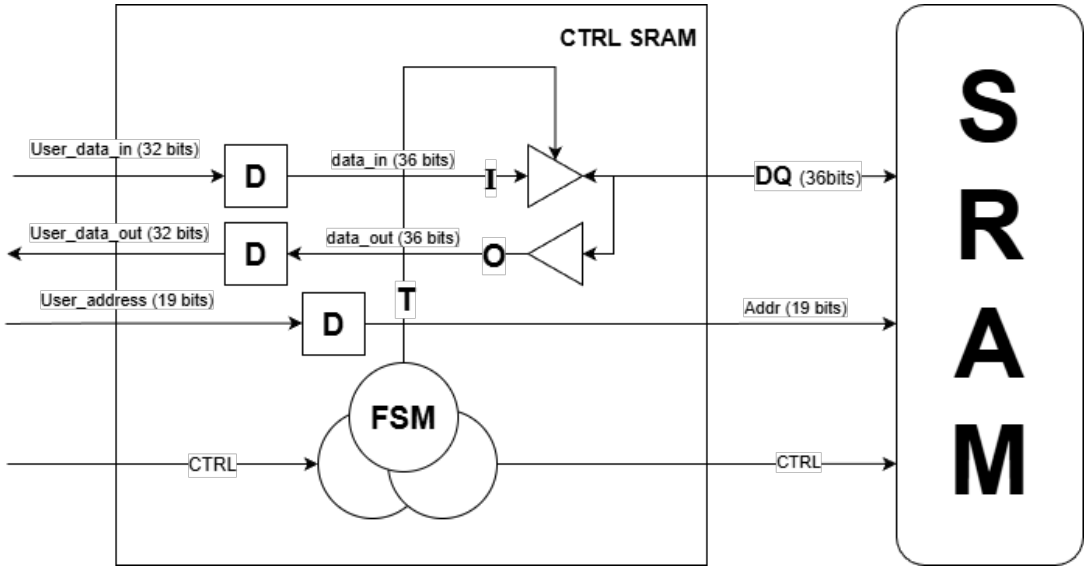

Figure 1.5: Controlleur SRAM

Coté utilisateur, Les signaux ont été ajustés pour tensir sur un nombre d'octets complet:

- `User_address` : Adresse de 19 bits
- `User_data_in` : Données d'entrée de 32 bits
- `User_data_out` : Données de sortie de 32 bits
- `Ctrl` : Signal d'écriture/lecture (1 bit)

Nous avons fait le choix de laisser l'adresse sur 19 bits pour pouvoir adresser toute la mémoire SRAM.
D'un point de vue utilisateur, nous aurions pu choisir de mettre l'addresse sur 16 bits pour avoir un nombrecomplet d'octet.

1.3.1 Finished State Machine

D'après la page 11 de la datasheet de la SRAM :

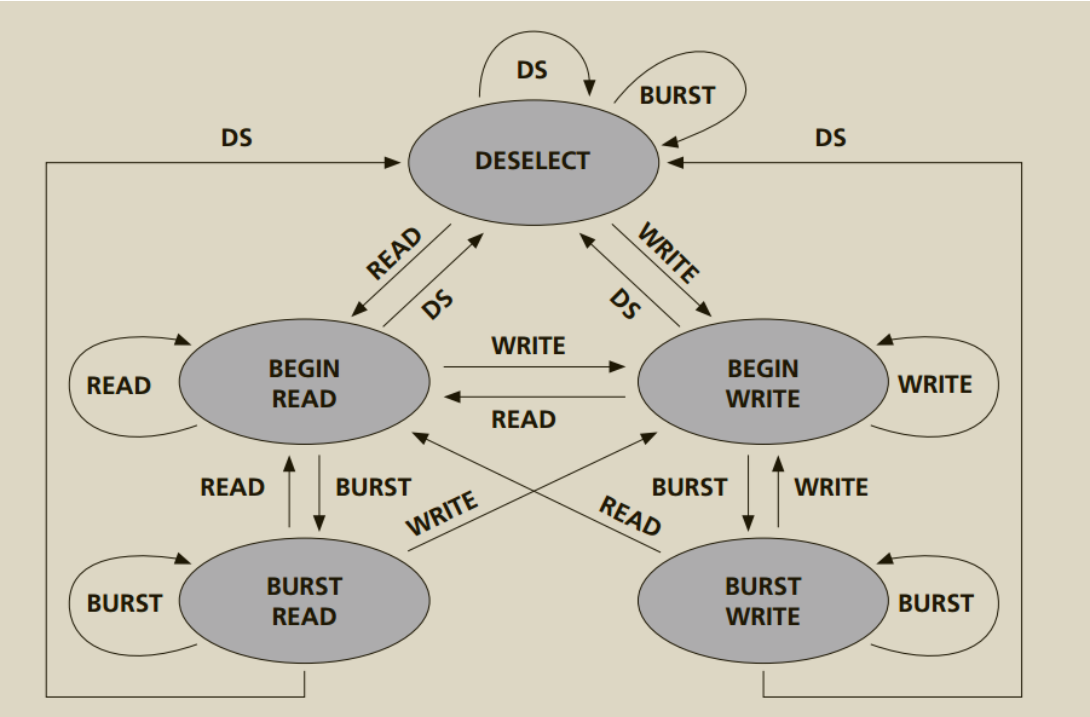

Figure 1.6: FSM

Pour le moment, nous n'allons pas implémenter le mode Burst.
Cela signifie que les signaux `MODE (LBO#)` et `ADV/LV#` seront à 0.

Implementation FSM

```
process(Clk,reset)
begin
if reset ='1' then
    state  <= INIT;

elsif Clk'EVENT and Clk = '1' then
    case state is
        When INIT =>
            state  <= IDLE;

        when IDLE =>
            if Ctrl ='1' AND Start ='1' then
                state <= READ;
            elsif Ctrl ='0' AND Start ='1' then
                state <= WRITE;
            else
                state <= IDLE;
            end if;

        when READ =>
            if Ctrl ='1' AND Start ='1' then
                state <= READ;
            elsif Ctrl ='0' AND Start ='1' then
                state <= WRITE;
```

```
            else
                state <= IDLE;
            end if;

        when WRITE =>
            if Ctrl ='1' AND Start ='1' then
                state <= READ;
            elsif Ctrl ='0' AND Start ='1' then
                state <= WRITE;
            else
                state <= IDLE;
            end if;
    end case;
end if;
end process;
```

1.3.2 Implementation du controlleur SRAM

D'après le sujet, plusieurs signaux seront à mettre à 0 :

- ZZ sooze mode : basse consommation. 1 pour activer
- nCKE : clock enable si mis à 0
- nBWA
- nBWB
- nBWC
- nBWD : Byte write enable. permet de selectionner quels octets sont écrits. mis à 0 pour écrire tous les octets.

Afin de creer un déclalge d'horloge entre la donnée utilisateur, l'adresse et la commande, nous allons utiliser des bascules D La manière la plus simple d'utiliser des bascules D est de assaigner une entrée à un signal interne du composant dans un processus sensible aux fronts de l'horloge.

Ici, sur front monant, nous allons décaler la donnée utilisateur de deux cycles d'horloge et recopier l'adresse utilisateur sur l'adresse de la SRAM.
Sur front descendant, nous allons ajouter les bits de parité à la donnée avant de l'envoyer à la SRAM et renvoyer la donnée lue de la SRAM vers la sortie utilisateur.

```
process(Clk)
begin
if Clk'EVENT and Clk = '1' then
--decalage de la donnée de deux fronts montant
decalage_data_in_1 <= User_Data_in ;
decalage_data_in_2 <= decalage_data_in_1;

Addr <= User_Address; -- recopier l'adresse d'entrée du controller en entrée de la SRAM
```

```
9  end if;
10 end process;
11
12 process(Clk)
13 begin
14 if Clk'EVENT and Clk = '0' then -- sur front descendant
15 Data_in_s <= "0000" & decalage_data_in_2; --Ajout des 4 bits de parités dans à la donnée
16 User_Data_out <= Data_out_s(31 downto 0); --Renvoie des 32 bits sur la sortie
17 end if;
18 end process;
```

1.3.3 Test bench

Le test bench de notre composant SRAM complet se résume à tester plusieurs points:

- Que les instructions de lecture/ecriture données par l'utilisateur son bien traitées par la FSM.
- Les données à écrire/lire sont bien traitées par l'IO Buffer.
- Les données sont correctements transmisent à la SRAM.

pour se faire, nous allons d'abord executer une sequence d'écriture/lecture à l'adresse 0. Ensuite, nous feront la même chose à l'adresse 7FFF... (soit la dernière adresse de la SRAM). Et enfin pour vérifier que la SRAM stock bien les données, nous vérifirons que la donnée à l'adresse 0 est toujours présente

Ces tests nous permettront de nous assurer du bon fonctionnement de notre implémentation du controlleur SRAM.

Le test bench est disponile sur GitHub sous le nom de fichier `tb_sram_controller.vhd`.

Voici sur la page suivante, la simulation du test bench:

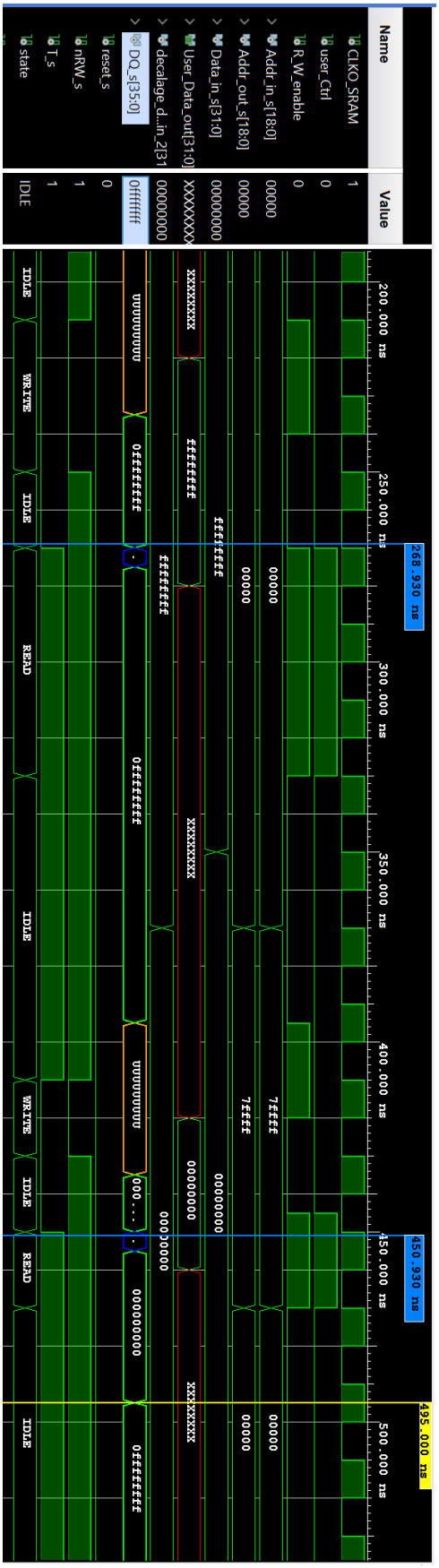

Figure 1.7: Simulation Test Bench Controlleur SRAM

Des marqueurs ont été placés pour indiquer les lectures aux différentes addresses. On constate que les données lues sont bien celles qui ont été écrites précédemment. à la lecture, le bus DQ de la SRAM est mis en haute impédance, ce qui est visible sur la simulation.

Les données et les adresses sont bien décalées d'un cycle d'horloge, comme prévu.

Ainsi, nous avons validé le bon fonctionnement de notre controlleur SRAM.

1.4 Conclusion

1.5 Remerciements

2 Annexe

2.1 Code-source / Document...

2.2 Bibliographie